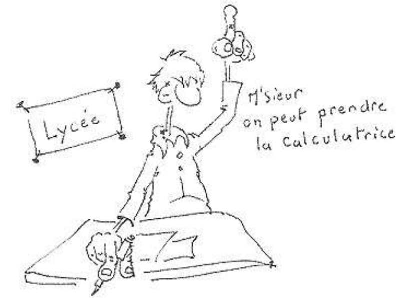
Nouvelle Aventure

XCAS



I - PROGRAMMATION

A - LE LANGAGE XCAS

Xcas, comme vous le savez, contient plus des centaines de fonctions pré-programmées. Évidemment, il est absolument impossible de les aborder toutes cette année. Le but de nos TD sera plutôt, dans un premier temps, de mettre en pratique quelques notions de base sur la programmation en prenant pour sujets des fonctions mathématiques simples : nous fabriquerons ainsi « à la main », de nombreuses fonctions qui sont pourtant déjà pré-définies en langage Xcas. Il s'agira en fait de découvrir, à travers des exercices concrets, la syntaxe Xcas et des algorithmes de base. Ceci nous permettra, dans un deuxième temps, d'utiliser les formidables capacités de Xcas en calcul formel pour travailler, sous un autre angle qu'en cours, les merveilleuses notions mathématiques que vous allez aborder en terminale S.

Un de vos fidèles compagnons pendant ces TD sera le `?`. En effet, en cas de doute sur une fonction Maple, il suffira de taper

```
> ?lafonction;
```

pour voir apparaître une aide. On peut sinon explorer les tutoriels où l'index au format html.

Quelques conseils avant de travailler.
Nous allons utiliser pas mal de variables que nous affecterons au hasard de nos travaux. On peut perdre le fil. Pour savoir quelles lettres sont déjà utilisées, on tape

```
> VARS()
```

ou on clique sur l'icône noire `var`. Pour désaffecter tout le monde, on tapera

```
> rm_all_vars()
```

ou on cliquera sur `Edit->Effacer->variables`. Pour désaffecter une seule variable, par exemple `"a"`, on tape

```
> purge(a)
```

.

De plus, on écrira nos lignes de programme sur l'éditeur de texte intégré en cliquant sur l'icône jaune `prg`. On testera le programme en cliquant sur `tester`. S'il n'y a pas d'erreur, on enregistre le programme dans un répertoire crée à l'avance et on calcule avec en allant dans l'historique via `->hist`.
Pour charger un programme déjà enregistré, on clique sur `load`. Enfin, voici la règle principale

Thèorème fondamental de l'algorithmique
Pour concevoir un programme, un crayon, une feuille et une gomme tu utiliseras

A - 1 : Sept manières de calculer 7!

1) Bien sûr, on peut commencer par faire

```
> 7!
```

mais le calcul de 7! n'est qu'un prétexte pour découvrir sur un exemple simple la syntaxe de programmation Xcas.

2) On pourrait alors taper

```
> 1*2*3*4*5*6*7
```

mais les choses pourraient se compliquer au moment de calculer 3232!

3) Nous allons donc utiliser une boucle *for* avec indice numérique : nous partons de 1, puis nous multiplions par 2, puis nous multiplions le produit précédent par 3, etc. Notons `p` le produit et `k` l'entier qui part de 1 et qui augment d'une unité jusqu'à atteindre 7. Cela donne

```
> p:=1; for( k=1; k<=7; k:=k+1){p:=p*k;}
```

4) Nous aurions pu créer la liste des entiers de 1 à 7 puis utiliser une boucle *for* indexée cette fois par les éléments de la liste. C'est plus compliqué, mais cela nous permet de découvrir comment Xcas traite les listes. Il faudra bien distinguer

▷ les **ensembles** (*set*) qui sont des collections « *non classées* » d'expressions toutes différentes séparées par des virgules et encadrées d'accolades.

```
> ens1:=set[2,4,1]; ens2:=set[2,5,8,5]
```

On peut effectuer les opérations usuelles sur les ensembles

```
> ens1 intersect ens2; ens1 union ens2; ens1 minus ens2;
```

l'ensemble vide se note

```
> set[]
```

▷ les **suites** (*sequence*) qui sont des collections « *classées* » d'expressions (c'est à dire avec un premier, un deuxième, etc.), différentes ou non, séparées par des virgules et encadrées ou non par des parenthèses.

```
> 5,7,5,1,2,3
```

On peut aussi utiliser les opérateurs `seq` et `$` pour des suites définies par une formule explicite

```
> seq(k^2,k=1..5)
> (p^2) $ (p=1..5)
> m $ 5
```

▷ les **listes** (*list*) qui sont des collections *classées* d'expressions séparées par des virgules et encadrées par des crochets. La différence, c'est qu'une suite, en tant que juxtaposition d'expressions, est en quelque sorte « en lecture seule », alors qu'une liste est une expression en elle-même et pourra donc « subir » des opérations algébriques.
Notez au passage quelques fonctions utiles

```
> s1:=(i) $ (i=-2..2); s2:=a,b,c,d,e;
> l1:=[s1]; l2:=[s2]; # une liste est une suite entre crochets#
> size(l2); # nombre d'opérandes #
> l2[3]; l2[0]; # notez bien que le premier opérande porte le numéro 0 #
> l2[2..4]
> select (x->x>0,l1); remove(x->x<0,l1);
>  subsop(l1,2=32); subsop(l2,'3=NULL'); # pour substituer ou supprimer un opérande #
> map(l2,cos);
> zip((x,y)->x*y,l1,l2);
```

Cela donne

```
> rm_all_vars()
```

```
> l:=[k $( k=1..7)]; p:=1; for( k:=1; k<=7; k:=k+1){p:=p*l[k];}
```

et on demande p dans l'historique.

5) Nous pouvons utiliser une boucle *while*

```
> rm_all_vars()
```

```
> p:=1; k:=1; while( k<7){k:=k+1; p:=p*k}
```

6) C'est bien beau, mais que ferons-nous quand il faudra calculer 32! ou 3232! et tous les autres ? Il faudrait créer un programme (on dira une **procédure**) qui donne $n!$ pour tout entier naturel n.

```
> rm_all_vars() # je ne vous le dirai plus#
```

```
> fact(n):={
local p,k; # nous aurons besoin de variables locales i.e. internes à la procédure #
p:=1;
for( k:=1; k<= n;k:=k+1){ p:=p*k ;}
} # termine la procédure #
```

```
> fact(32)
```

7) Le meilleur pour la fin : la **procédure récursive**, qui s'appelle elle-même

```
> factr(n):={
if (n==0) 1; # un test if...else pour régler le cas de 0! #
else n*factr(n-1);
}
```

```
> factr(32);
```

En fait, ce mécanisme correspond à une suite numérique qui s'écrirait mathématiquement $u_n = n \times u_{n-1}$ et économise beaucoup de mémoire car elle repart du dernier résultat calculé.

A - 2 : À vous de jouer...

Exercice 1 Partie entière
Déterminez une procédure `E(x)` qui, à un réel positif x, associe sa partie entière.

Exercice 2 Valeur absolue
Déterminez une procédure `ab(x)` qui, à un réel x, associe sa valeur absolue.

Exercice 3 Moyenne
Déterminez une procédure `moy(l)` permettant de calculer la moyenne des éléments d'une liste `l` de nombres réels. Il faut commencer par calculer la somme des opérandes de la liste puis diviser par le nombre d'opérandes de la liste.

Exercice 4 Somme
Vous savez peut-être que la suite $(S_n)_{n\in\mathbb{N}}$ de terme général

$$S_n = \sum_{k=0}^{n} \frac{1}{k!}$$

est croissante et converge vers e. Nous l'admettrons dans cet exercice.

1) Déterminez une procédure `S(n)` qui, à un entier naturel n, associe S_n.
2) Déterminez une procédure `seuil(p)` qui, à un entier naturel p, associe le plus petit entier naturel n tel que $|S_n - e| \leqslant 10^{-p}$. Vous aurez besoin de savoir que e se dit `exp(1)` en Xcas.

Exercice 5 Équation du second degré
Déterminez une procédure `sol(a,b,c)` qui, à une équation $ax^2 + bx + c = 0$, associe son ensemble des solutions.

Exercice 6 test sur une liste
Déterminez une procédure `test(l)`, `l` étant une liste d'entiers, qui teste si ses éléments forment une suite croissante.

Exercice 7 Étude des branches infinies
Nous allons écrire un programme étudiant les branches infinies d'une fonction numérique. Voici un bref rappel de cours : si $f(x)/x$ admet une limite finie a en l'infini et si $f(x) - ax$ admet une limite finie b, alors $\mathcal{C}_f$ admet la droite D d'équation $y = ax + b$ comme asymptote au voisinage de l'infini.
Il va falloir utiliser pas mal de tests if. Il va d'abord falloir demander à XCAS de calculer des limites. Il sait faire :

```
> limit(sin(x)/x,x=0); limit(1/sqrt(x),x=0); limit(sin(x)/x,x=+infinity); limit(sin(x),x=+infinity);
```

Ensuite, il va falloir tester la réponse : a est réel ou infini ? La limite existe ou n'existe pas ?
Nous allons maintenant construire une procédure `branche(f)` qui renvoie l'éventuelle asymptote de la courbe $\mathcal{C}_f$. Elle contiendra deux variables locales a et b. Nous utiliserons l'instruction `return` qui quitte la procédure et évite les `else`. Je vous livre le début

```
> branche(f):={

> local a,b;

> a:=limit(f(x)/x,x=+infinity):

> if (a==undef) return "pas d'asymptote";
⋮
return "asymptote d'equation y=("+a+")*x+"+b;
}
```

À vous d'imaginer ce qui manque...
On rentrera ensuite par exemple `branche(x->(x^3+9)/(x^2-1))`

Exercice 8 Autres pistes...
Á vous d'imaginer un programme qui étudie la dérivabilité d'une fonction f en un point, le sens de variation d'une fonction, la résolution de l'équation $x^3 + px + q$ à la Bombelli-Cardano, etc.